UNIVERSIDAD AUTONOMA DE MADRID
ESCUELA POLITECNICA SUPERIOR
Grado en Ingeniería Informática

TRABAJO FIN DE GRADO

Aplicación Web de Gestión de Equipos de Fútbol

Raúl Palomares Gil

Tutor: Daniel Hernández Lobato

Julio 2015

Resumen

En este trabajo de fin de grado se ha decidido llevar a cabo una aplicación web de gestión de equipos de fútbol base, más orientado a clubes pequeños, donde la organización de los equipos y el seguimiento de los jugadores se suele hacer de forma más rudimentaria y menos esquematizada.

El creciente interés por el fútbol hace que se reafirme como el deporte más popular en nuestro país, y cada vez sean más los niños que opten por este deporte como actividad extraescolar. Esto sumado a la cantidad de equipos de colegios, y clubes que nacen en los barrios, deja un hueco en el mercado con un enorme número de clientes potenciales y grandes perspectivas de crecimiento.

Este hueco es exactamente el tratamiento y visualización de toda la información referente a estos clubes poco profesionalizados, y es por eso que esta herramienta ayudará a mejorar tanto el aspecto deportivo como el económico.

Para la realización de este proyecto se han definido unos requisitos acordes a las optimizaciones citadas, para ello se ha contado con la ayuda de terceras personas relacionadas de forma directa con estos equipos de fútbol base, que sabían de primera mano las necesidades exactas que podría solventar una aplicación.

El siguiente documento recoge una breve descripción de la aplicación, hasta donde se ha querido profundizar, así como un estudio más exhaustivo de las etapas previas al desarrollo del proyecto, como se ha llevado a cabo este desarrollo y posibles mejoras que harían de esta aplicación, una herramienta mucho más útil.

Palabras Clave

Java, JavaServer Faces, PrimeFaces, software para equipos de fútbol, Modelo Vista Controlador, Aplicación web.

Abstract

The aim of this final degree project is to develop a little league football team management web application, focused on small clubs, where its teams organization and the monitoring of its player, are used to be done in an old-fashioned way and not properly enough.

The rising interest in football, has made it the most popular sport in our country, and through the past of time, football still the option more fecuently choosen by kids as their extracurricular activity. If we add to the mentioned reasons, the fact that the there is a high amount of school teams, and little neighbourhood teams, we can rigorously affirm that exist a gap in the market, with an enormous amount of potential clients and rather high growing perspective.

Going further, we can define this gap as the treatment and visualization of the whole information relative to this low professionalized clubs, hence this application will help to improve the results hoped in all about sports concern, as well as in economics does.

In order to accomplish the goals of this project, some requirements has been settled. To do that, it is fair to recognize the help of external people who are directly related with these little leagues football teams, using their knowledge at the mentioned area to extract their most relevants needs.

The next document sumarizes a brief description of this project, as well as an exhaustive description of the previous steps that took place before the development, how this development was done and potential improvements that will make it a more useful tool.

Key words

Java, JavaServer Faces, PrimeFaces, football team software, Model Vew Controlller, web application.

Nomenclatura

API	Conjunto de métodos y procedimientos que ofrece una biblioteca.
Bean	Componente software que tiene la particularidad de ser reutilizable y así evitar la tediosa tarea de programar los distintos componentes uno a uno.
Connection Pool	Colección de conexiones abiertas a una base de datos de manera que puedan ser reutilizadas al realizar múltiples consultas o actualizaciones.
Framework	Conjunto estandarizado de conceptos, prácticas y criterios para enfocar un tipo de problemática particular que sirve como referencia, para enfrentar y resolver nuevos problemas de índole similar.
Hosting	Servicio que provee a los usuarios de Internet un sistema para poder almacenar cualquier contenido accesible vía web.
HTML	Lenguaje de programación de marcado para la elaboración de páginas web.
JAVA	Lenguaje de programación de propósito general, concurrente, orientado a objetos que fue diseñado específicamente para tener tan pocas dependencias de implementación como fuera posible.
Java EE	Es una plataforma de programación para desarrollar y ejecutar software de aplicaciones en el lenguaje de programación Java.
MVC	Modelo Vista Controlador.
SQL	Lenguaje declarativo de acceso a bases de datos relacionales que permite especificar diversos tipos de operaciones en ellas.

Índice

Índice de figuras

1. INTRODUCCIÓN

1.1. Motivación

Hoy en día y desde hace ya bastantes años, las aplicaciones web son parte fundamental de nuestra vida. Casi sin darnos cuenta hemos pasado a tener una gran dependencia de ellas, ya sea para pedir cita en el médico, revisar nuestro expediente académico o hacer la compra. Debido a su gran utilidad y sobretodo a la posibilidad que nos brindan de realizar una acción prácticamente desde cualquier lugar, se han convertido en una herramienta que sencillamente, nos facilita la vida.

A todo esto le tenemos que sumar sus beneficios a un nivel no funcional. Este tipo de aplicaciones no tienen problemas de compatibilidad, basta tener un navegador actualizado para poder utilizarlas, ni nos consumen memoria en nuestros dispositivos. A los beneficios de las aplicaciones web, hay que sumarle los de las tecnologías móviles, que van adquiriendo una grandísima importancia en nuestras vidas, prueba de ello son los Smartphone. Tienen un carácter complementario con respecto a las aplicaciones web. Gracias a mecanismos ofrecidos por algunas plataformas de desarrollo, la portabilidad de las aplicaciones web a las tecnologías móviles requiere un esfuerzo muy pequeño para los desarrolladores y una ventaja muy grande para los usuarios.

Actualmente el fútbol es el deporte más popular en nuestro país. En nuestra ciudad existe un elevado número de equipos de fútbol base, los cuales encajan con el patrón de una organización, ya sean, clubes deportivos, asociaciones deportivas o sociedades. Estas organizaciones suelen contar con un elevado número de equipos, lo que conlleva un número más elevado aún de deportistas y gente relacionada con la gestión del club, así como datos relevantes con la evolución del mismo ya sean partidos o entrenamientos. A parte de todo esto, se suele dar en estas organizaciones el denominador común de un presupuesto ajustado.

Para optimizar la gestión, tanto deportiva como económica, se necesita recoger toda la información relacionada al respecto, y mostrarla, tras ser procesada, de manera organizada a los usuarios. Es ahí donde surge la motivación de este Trabajo de Fin de Grado.

A pesar de que existen plataformas parecidas, se busca hacer una aplicación básica y genérica, que pueda ser fácilmente ampliable por módulos según los requerimientos específicos de cada cliente.

En definitiva, se busca mejorar la gestión de equipos de fútbol base, de tal forma que

con el procesamiento de datos y la organización de la información se reduzcan gastos y se incrementen beneficios, así como se establezca un mayor control de la actividades deportivas para mejorar sus resultados.

1.2. Objetivos

El mayor objetivo de este Trabajo de Fin de Grado es sin duda saber aplicar los conceptos adquiridos a lo largo de la carrera en un proyecto real. Para ello es necesario tener claro los conceptos tanto de "Proyecto de Análisis y Diseño de Software" como de "Sistemas Informáticos", pero hay que apoyarse en algunos conocimientos adquiridos en otras materias como los conceptos de planificación de un proyecto estudiados en "Ingeniería del Software" o todo lo referente a las bases de datos visto en "Estructura de Datos".

Con un proyecto de estas características no solo se aprende como se desarrolla una aplicación web a grandes rasgos, sino que se ha de estudiar minuciosamente cada paso del ciclo de vida del software para ver que decisiones se toman y como afectarán a futuras fases. Uno de los objetivos más bonitos que he perseguido en el todo el ciclo de vida del proyecto, es un buen diseño. El estudio del diseño de la aplicación evita posibles fallos futuros, que tengan un enorme coste de tiempo y es, a mi juicio, donde más hincapié hay que poner. Para ello, previamente se ha tenido que investigar acerca de la materia sobre la que se está trabajando, y, aunque no ha existido ningún cliente concreto de donde se hayan sacado unos requisitos específicos, por medio de ese estudio se ha sabido conocer las necesidades de una entidad como las descritas anteriormente.

En cuanto al desarrollo, el reto más importante para la realización del proyecto es aprender a mezclar las tecnologías web, y cuando usar cada una. Aunque tener una base solida de programación en Java es fundamental (usando el framework que he usado), también hay que aprender el funcionamiento de AJAX, JavaScript y SQL entre otros.

Finalmente se ha buscado, no solo aprender cosas nuevas y afianzar conceptos pasados, sino realizar un proyecto con el que sentirse orgulloso con uno mismo por el trabajo realizado, y tener la ilusión de que pueda ayudar a otras personas en mayor o menor grado. Al ser un proyecto propio, he contado con un extra de motivación y he perseguido dar mayor peso a la toma de decisiones con la ayuda del consejo de mi tutor.

1.3. Estructura de la Memoria

La estructura principal de la memoria está compuesta por seis puntos, que a su vez, están compuestas por otras subsecciones que detallan más específicamente el tema que estén tratando. Estas secciones principales son:

- Estado del arte: Se hace un breve estudio sobre algunas de las tecnologías actuales más relevantes en el ámbito de desarrollo de aplicaciones web y se justifica por qué se ha elegido la tecnología usada.
- Análisis: En este apartado se recogen los requisitos que debe a tener la aplicación, tanto funcionales como no funcionales, así como se estudia la elección del ciclo de vida que se va a emplear.
- Diseño y desarrollo: Se busca explicar las características del framework utilizado y la arquitectura usada para desarrollar la aplicación.
- Pruebas y resultados: Se detallan las pruebas realizadas.
- Incrementos futuros y conclusiones: Descripción de el como y el porque de las mejoras propuestas para el futuro y balance total del trabajo realizado.

2. ESTADO DEL ARTE

2.1. Comparación de frameworks

En este capitulo se lleva a cabo una pequeña investigación con el fin de optimizar la fase de desarrollo del proyecto. Esta investigación consiste en una comparativa entre algunos de los frameworks más utilizados actualmente para realizar aplicaciones web, donde se detallan sus características, puntos fuertes y puntos flojos. Con esto, no se pretende sacar una conclusión absoluta sobre que framework es mejor, ni más eficiente, tan solo se pretende justificar la elección de la tecnología usada para el desarrollo de este proyecto en concreto.

Para llevar a cabo esta investigación se ha estudiado a fondo bibliografía basada en cada uno de estos frameworks, de donde se ha obtenido la información relevante a ellos. Algunos de estos frameworks ya se conocían de la carrera, lo que ha facilitado esta tarea, otros en cambio, se han estudiado desde cero. Además se ha procurado coger los framewokrs más relevantes de cada lenguaje de programación, en lugar de coger varios de un mismo lenguaje.

2.1.1. DJANGO (Python)

Esta basado en el lenguaje Python, y por ello, cuenta con sus principales ventajas, como lo son la escritura de código bastante fácil de entender, y el desarrollo de aplicaciones rápidas y potentes.

La vista en Django se edita mediante el uso de plantillas, las cuales se usan para generar HTML, con etiquetas cuyo significado van desde acciones, a variables.

Ventajas:

- Su punto más fuerte es la velocidad y su simpleza a la hora de programar. Está estructurado de tal manera que sus aplicaciones web se crean muy rápidas.

- Tiene una interfaz automática de administración de la aplicación web que se esté creando. Se trata de una interfaz web limitada a administradores del sitio, y que permite añadir, editar y eliminar contenidos. El objetivo de esto es evitar crear de nuevo algo, que es repetitivo. Esta característica funciona mediante la lectura de meta-datos de su modelo, para así ofrecer una interfaz potente y lista para usar.

- Su orm (object relational mapping), Django utiliza un modelo para ejecutar código SQL y devuelve estructuras de datos en Python que representan las filas de las tablas de la base de datos que se esté usando. Esto permite mantener todos los elementos dentro del mismo lenguaje, manteniendo la productividad elevada al trabajar en un único entorno de programación. Este punto en algunas ocasiones puede traer problemas, ya que se puede dar el caso de el código en Python no se sincronice adecuadamente con el contenido de la base de datos.

- Seguridad, Django controla algunos tipos de ataques a las aplicaciones web gracias a su diseño. Algunos de los problemas más conocidos que Django solventa automáticamente son el phishing, la falsificación de sesión o la inyección sql.

Desventajas:

- Alta curva de aprendizaje.

- La mayoría de servidores no tienen Python, y si lo tienen, la configuración es difícil.

- Aparenta implementar el patrón MVC, pero el controlador es llamado vista (que datos serán mostrados), y la vista "template" (como serán mostrados estos datos). Esto en ocasiones puede resultar algo lioso si se está acostumbrado a desarrollar usando MVC.

2.1.2. JavaServer Faces (Java)

Framework para desarrollar aplicaciones basadas en Java. Simplifica el desarrollo en aplicaciones Java EE. La vista del framework está basada en xhtml, sintaxis muy parecida a html pero con la peculiaridad de las etiquetas propias de JavaServer Faces (una versión avanzada del lenguaje de expresiones de JSP) con las que se transmite la información de la vista al modelo y viceversa.

En cuanto al *modelo*, se caracteriza por el uso de "beans". Estos "beans" son clases con un conjunto de atributos y métodos getter y setter que devuelven y actualizan sus valores.

La conexión de la vista con el *controlador* se hacen por medio de los denominados Managed Beans, los cuales toman valor en la vista para ser usados en la lógica, o la lógica les proporcionan un valor para que se muestren en la vista.

Las conexiones a bases de datos se pueden hacer de varias maneras, una de ellas es mediante el uso de JDBC connection pool, que, tras una configuración previa, tienen un fácil manejo por parte del desarrollador para acceder a las bases de datos. Otra forma de establecer conexiones a bases de datos es mediante plataformas Java (por ejemplo Hibernate), la cual facilita todavía más el mapeo de atributos entre una base de datos relacional tradicional y el modelo de objetos de una aplicación.

Ventajas:

- Permite separar fácilmente la vista y la lógica.
- Fácil manejo de las API's para representar componentes en la interfaz de usuario y administrar su estado, manejar eventos... Así como la posibilidad de crear elementos más complejos en la vista, de una manera más sencilla mediante el uso de librerías.
- Un modelo de eventos del lado del servidor.
- Fácil aprendizaje, sin necesidad de conocer el Framework completamente para usarlo.
- Existen etiquetas por parte de la vista que facilitan enormemente el uso de Ajax.

Desventajas:

- Poca flexibilidad de los componentes propios en la vista, y menos aun si esos componentes provienen de una librería.

Como podemos ver en la siguiente comparativa de frameworks Java, JavaServer Faces es el segundo framework más utilizado por detrás de Spring MVC. Sin querer introducirnos mucho en Spring MVC, solo comentar que se ha preferido el uso de JavaServer Faces, a parte de por ser parte de los standarts Java EE, por el hecho de contar con numerosas librerías (PrimeFaces, RichFaces...) que facilitán la creación de una interfaz más lograda.

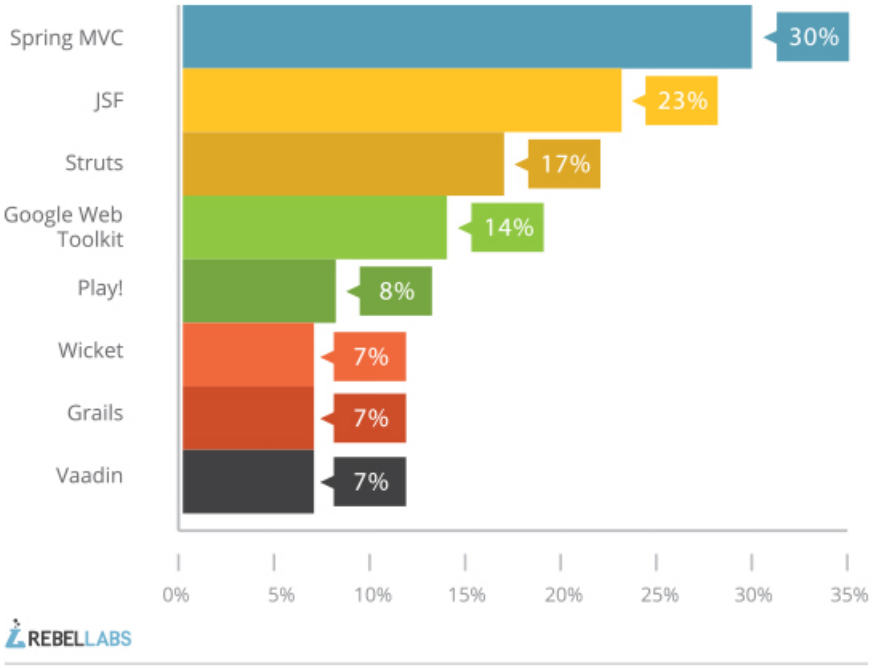

Figura 1: COMPARATIVA FRAMEWROKS DE JAVA

2.1.3. Symfony (PHP)

Framework basado en php para desarrollar aplicaciones web. Basado en la arquitectura MVC, cuenta con las características de este lenguaje de programación, como su facilidad a la hora de aprender, es software libre, es un lenguaje seguro por ser invisible al navegador web, uso de métodos mágicos o la posibilidad de programar orientado a objetos. Como requisito especial de este framework, es que está escrito en php5, y se recomienda el aprendizaje de esta versión de php y no de las anteriores. La vista de este framework se desarrolla en HTML, facilitando la tarea a los desarrolladores, que pueden editar la vista sin necesidad de conocer el framework. También incluye un lenguaje para crear plantillas llamado Twig fácil de leer para diseñadores web.

En cuanto al controlador, se divide en varios componentes:

Front controller: Actua como un controlador tradicional, obtiene la entrada y determina la acción a configurar.

Actions: Contiene la logica, comprueba el request y prepara los datos para mostrar en la salida.

Request, response y objetos de session: Dan acceso a los parametros "request" y a la información del usuario actual.

Filters: Se ejecutan despues de cada "request" a modo de capa de seguridad, es extendible a otras plataformas.

Ventajas:

- Fácil de instalar y configurar en la mayoría de las plataformas.
- Independiente del sistema gestor de bases de datos. Las bases de datos son relacionales, y para acceder a ellas de forma más eficiente y usando la orientación a objetos, se usa una interfaz ORM (Oriented-Relational Mapping), la cual traduce un objeto en información para la base de datos, y viceversa. Se usan objetos en lugar de registros y clases en vez de tablas. El punto realmente fuerte de la interfaz ORM es la portabilidad, ya que se puede modificar el sistema de datos sin necesidad de cambiar la lógica. Por ejemplo, se puede empezar a desarrollar la aplicación en SQLite, y cambiarla a MySQL, PostgreSQl o Oracle cuando el cliente lo decida tan solo cambiando una linea de la configuración.
- Soporte de email incluido así como otras APIs ya desarrolladas (como por ejemplo OpenSocial, que permite intercambio de información en redes sociales).

Desventajas:

- Cuesta al principio.
- Gran parte de la velocidad de Symfony se debe al uso extensivo de la memoria caché.
- La gran flexibilidad de este framework a la hora de desarrollar hace que si la aplicación no la desarrolla un experto en php, provoque errores leves como un incremento en la memoria utilizada o un decremento de la velocidad.
- A pesar de tener una función especifica editable que provoca dinamismo del lado del cliente sin necesidad de refrescar la página, no posee una etiqueta específica para hacer esto, y la función previamente explicada no es del todo intuitiva.

2.1.4. RUBY ON RAILS (Ruby)

Es uno de los proyectos de código abierto más importante de los últimos tiempos, está basado en el lenguaje Ruby y sigue la arquitectura MVC.

Aplica tanto para la capa de presentación (vista), como para la lógica y el acceso a datos. Al igual que JSF, Php o Asp, el código Ruby en la vista está embebido en el código HTML, de manera que no se logra una separación total entre el código HTML y el código de la implementación. La lógica de la aplicación se ejecutan en código Ruby.

Ventajas:

- Metaprogramacion: Mientras que otros marcos de trabajo usan generación de código (que aumenta la productividad de los usuarios) y scripts para personalizar código, la metaprogramación sustituye estas dos técnicas primitivas y suprime sus desventajas. Ruby es uno de los mejores lenguajes de metaprogramación.

- Active Record: Los usuarios manipulan las tablas de una base de datos a través de objetos Esto permite diseños y conexiones sencillas entre bases de datos y objetos de la aplicación. Definiendo las tablas y ejecutando unos pocos scripts es posible tener una aplicación totalmente funcional que realice las operaciones básicas. Es el punto más fuerte de Ruby on Rails.

- Ficheros de configuración: Algo poco atractivo sobre marcos de trabajo para .NET o Java es, en ciertas ocasiones, tener que editar ficheros de configuración poco intuitivos. Rails no posee ficheros de configuración (posee uno pero es para temas generales, como la dirección de la base de datos).

- Scaffolding: A menudo se crea código temporal en los primeros pasos del desarrollo para crear rápidamente una aplicación, y poder ver como funcionan juntos los componentes principales. Rail crea automáticamente gran parte de este código.

Desventajas:

- No existe un framework de GUI (Graphical User Interface) multi-plataforma ampliamente aceptada.
- Poco apoyo de la comunidad Ruby.

2.1.5. ASP.NET (.NET)

Frmaework de Microsoft para aplicaciones web, tiene la característica de aceptar cualquier lenguaje admitido por el .NET framework. Alguno de estos lenguajes son .Net, C#, Visual Basic, J#, Perl o Cobol entre los más conocidos. La arquitectura que se use dependerá sobretodo del lenguaje con el que se desarrolle la aplicación, aunque la arquitectura MVC es muy usada. Otra arquitectura bastante conocida es el modelo Code-Behind, el cual Microsoft recomienda para realizar programación dinámica, esta arquitectura consiste en separar la presentación del contenido en vez de hacer una programación lineal permitiendo al diseñador web a enfocarse en el diseño y no alterar el código de programación.

Otra particularidad de este framework son los servicios COM+ los cuales se trata de una manera de hacer componentes reutilizables y seguros, entre otras cosas. De los servicios que ofrece COM+, hay que destacar:

CRM (Compensating Resource Management): Permite la aplicación de propiedades propias a componentes, como la durabilidad.

JIT (Just-In-Time activation): Permite la carga de un componente a memoria solo y durante la petición de un método por parte COM+ por parte del usuario. Este mecanismo elimina la instancia del objeto y libera memoria tan pronto como el componente sea utilizado por el usuario.

Object pooling: Su funcionalidad deriva del Connection pooling. Permite guardar instancias de objetos, de manera que en cualquier petición a estos, la creación del objeto sea más rápida.

Sincronización: Administra objetos que utilizan varios hilos de ejecución.

Seguridad: Servicios que protegen el acceso a recursos críticos.

Las *vistas* de las páginas creadas en ASP dependerán del lenguaje en el que sean desarrolladas, por ejemplo, en caso de .NET, la vista se crea con etiquetas HTML o XHTML, y otras etiquetas que definen la unión con la lógica de la aplicación. También se pueden crear vistas en lenguajes como C# o Visual Basic.

Respecto a la lógica de una aplicación, no presenta muchas diferencias con otros frameworks a parte del uso de servicios COM+ ya comentados con anterioridad.

Tiene un conjunto de extensiones para implementar AJAX. Contiene una librería Microsoft AJAX Library que facilitan el desarrollo al programador. De esta manera, en la vista, se puede usar esta tecnología mediante el uso de etiquetas especificas. Suelen ser intuitivas y fácil de usar.

Las conexiones a bases de datos se llevan a cabo mediante cadenas de conexión almacenadas en un archivo de configuración. Las llamadas a bases de datos se hacen con métodos, y sin lenguaje sql, algo positivo para el desarrollador. También existen plataformas para abstraer todavía más el uso de bases de datos, como por ejemplo, Hibernate.

Ventajas:

- Orientado a objetos
- Flexibilidad de lenguaje. Permite disfrutar de las ventajas de ASP sin necesidad de ser programado en .NET.
- El entorno de desarrollo configura bastantes parámetros. Esto ayuda bastante al desarrollador.
- Funciona tanto para servidores propios de Microsoft, como para Apache.
- Facil uso de AJAX.

Desventajas:

- Puede haber errores al usar un servidor Linux en vez de Windows
- No tiene mecanismo de invocación remota de sus métodos o funciones.

2.1.6. Apache Cocoon (Java)

Framework basado en Java. Tiene bastantes diferencias con otros Frameworks basados en este lenguaje. Se basa en un modelo de componentes (modelo) y en el concepto

de tuberías de componentes (controlador). Cada componente se especializa en una operación en particular y el contenido ingresa en la tubería y es modificado por las sucesivas etapas hasta generar la respuesta que es enviada al cliente. Tiene la característica de un archivo llamado "sitemap" en el cuál se especifican los componentes, las vistas, los recursos, los conjuntos de acciones y las tuberías que tendrá la aplicación. Este archivo se configura a través de un archivo xml. Incluye un "pool de conexiones" a base de datos, que permite reutilizar un conjunto de conexiones a la base de datos en vez de tener que abrir y cerrar conexiones cada vez que se hace una conexión al servidor. También detecta el cliente y puede servir diferentes contenidos dependiendo de él. La vista, está provista de etiquetas para el uso de ajax, de esta manera se puede dar fácilmente dinamismo a la parte del cliente.

Ventajas:

- Software libre, ya que está hecho por Apache project.
- Permite separar claramente el contenido de la presentación y de la lógica.
- Permite modificar el comportamiento de la aplicación sin conocer el lenguaje en el que se está implementando (Gracias al "sitemap"). Este es sin duda su punto más fuerte.

Desventajas:

- Diseño gráfico en xsl (requiere amplio conocimiento de xsl). En versiones posteriores, el lenguaje pasa a ser xsp (una forma dinámica de xml).
- Comunidad de desarrolladores pequeña.
- Curva de aprendizaje elevada.

2.2. Conclusión

Tras valorar las ventajas y desventajas de cada framework para el desarrollo de la aplicación, y debido a sus cualidades, se ha elegido JavaServer Faces como marco para realizar la aplicación. A pesar de todas las ventajas descritas anteriormente, los puntos más fuertes de esta tecnología que me motivaron a escogerla para crear la aplicación fueron los que se describen a continuación.

- Su mayor virtud, sin ninguna duda, es que JavaServer Faces permite aprovechar todas las ventajas del estándar Java EE. Un ejemplo del beneficio de Java EE es el uso de la API Enterprise Java Beans, la cual facilita la construcción de aplicaciones. El funcionamiento de la API es el siguiente; los servidores de aplicaciones proveen objetos desde el lado del servidor al cliente, a estos objetos se les denomina Enterprise Java Beans (EJB) y contienen la lógica de la aplicación. La ventaja que proporcionan los EJB al desarrollador es que permite a este centrarse únicamente en desarrollar la lógica abstrayéndose de problemas como la concurrencia o la seguridad, que se desarrollan a parte. De este modo los EJB se convierten en objetos flexibles y reutilizables.

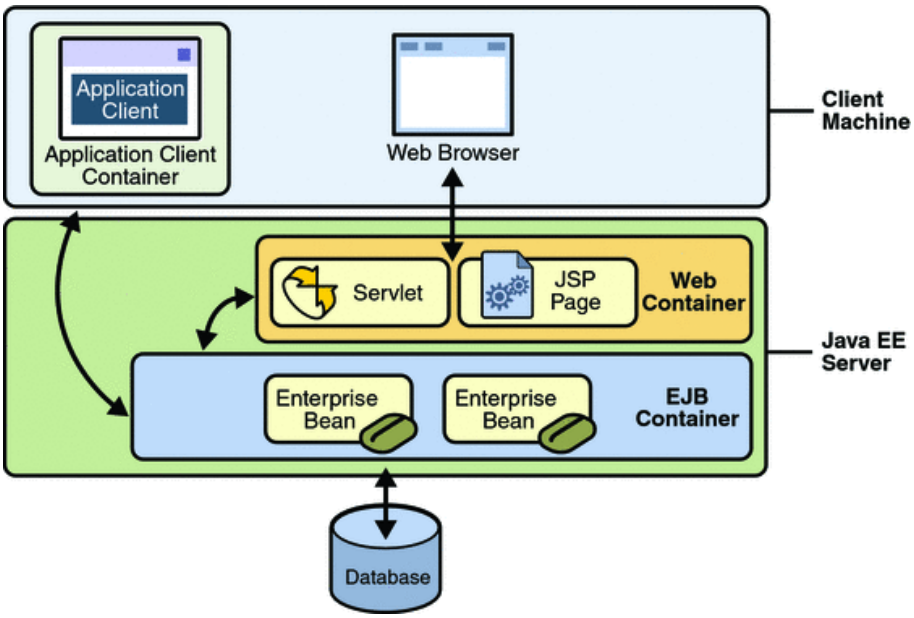

Figura 2: ARQUITECTURA EJB

La segunda gran ventaja de los EJB y quizás la más importante, es que pueden soportar invocación remota. De esta forma, los EJB facilitan enormemente el desarrollo, por ejemplo, de una aplicación móvil, donde el modelo de la aplicación no haría falta volver a desarrollarlo, sino que sería reutilizado mediante el uso de estos EJB.

- Otro punto fuerte de esta tecnología es la facilidad para usar AJAX. Gracias a la tecnología XHTML que usa JavaServer Faces para realizar la vista de la aplicación, es relativamente sencillo integrar AJAX en la aplicación mediante el

uso de etiquetas bastante intuitivas. De esto se hablará más adelante. Del mismo modo, los ficheros XHTML también me permiten integrar etiquetas especiales que definen componentes JavaServer Faces.

- Finalmente, decir que también me ha llevado a esta elección la facilidad que supone poder desarrollar la mayoría de la aplicación (modelo y controlador) en lenguaje Java, lenguaje que conozco perfectamente, que posee gran cantidad de librerías y facilidades y que además tiene detrás una inmenso número de desarrolladores que facilitan la resolución de cualquier problema.

2.3. Tecnologías usadas

Para acabar este apartado, se describen las tecnologías usadas durante todo el desarrollo del proyecto, muchas de ellas han sido novedosas.

NETBEANS Es un entorno de desarrollo libre compuesto por herramientas de programación. Se puede descargar paquetes adaptados a las necesidades específicas de cada desarrollo, en este caso, se ha utilizado NetBeans IDE Bundle for Web & Java EE. Este paquete incluye la tecnología J EE, así como el servidor GlassFish.

JSF Ya se ha comentado esto previamente las características de esta tecnología.

PRIMEFACES Se trata de una librería de componentes para JavaServer Faces que facilita la creación de las vistas de una aplicación web. Consta de una infinidad de etiquetas, las cuales ahorran trabajo al desarrollador, y sobre todo, soporta AJAX y es compatible con móviles.

CSS CSS (Cascading Style Sheets) es un lenguaje de programación que se emplea para dar estilo a los documentos escritos en HTML, así como a los documentos escritos en sus lenguajes variantes (como XHTML).

JAVASCRIPT Es un lenguaje de programación interpretado que se utiliza para crear webs mejoradas en cuanto a la interfaz de usuario y a su dinamismo, ya que se utiliza del lado del cliente. No tiene acceso al lado del servidor, por lo que su uso es únicamente visual.

AJAX El lenguaje AJAX (Asynchronous JavaScript And Xml) es un lenguaje de código abierto que proporciona una gran ventaja para crear una web dinámica y rápida. Se usa para crear aplicaciones interactivas que se ejecutan del lado del cliente y evitan que se tenga que refrescar la página cada vez que el usuario interactúe con esta.

POSTGRESQL Es un sistema de gestión de bases de datos relacionales de código abierto. Una de las alternativas a productos comerciales más comunes debido a su funcionalidad básica y seguridad.

TORA Es un conjunto multiplataforma de herramientas de código abierto que proporciona soporte para sistemas de bases de datos (entre ellos PostgreSQL) y sirve para gestionar una base de datos de manera muy sencilla y eficaz.

GLASSFISH Servidor de aplicaciones web de código abierto que implementa las tecnologías definidas en J EE, permitiendo ejecutar aplicaciones portables y escalables.

LATEX Sistema de edición de texto orientado a la creación de documentos escritos de alta calidad tipográfica. El texto se configura mediante un conjunto de etiquetas que indica las características de este, dando amplia libertad al usuario para crear el texto exactamente como el quiera, evitando así problemas que suelen dar los editores más comunes. Es software libre.

3. ANALISIS

En este apartado se va a proceder a la mostrar los requisitos educidos con el objetivo de intentar clarificar los objetivos de la aplicación lo máximo posible. Esto tendrá lugar en el apartado de requisitos funcionales, los cuales están organizados por las secciones en las que se divide la aplicación. También se analizarán los requisitos no funcionales que se necesitarán dados los requisitos funcionales y las características de las tecnologías usadas.

3.1. Requisitos funcionales

RF1.Autenticación(Log-In)

Para que el usuario pueda acceder a la aplicación será necesario que este introduzca su nombre y contraseña. La aplicación comprobará en la base de datos si existe el nombre, y la contraseña correspondiente es correcta, en tal caso el usuario accederá a la aplicación. En caso contrario, se mostrará un mensaje de error y no se podrá acceder.

RF2.Cierre de sesión

Para que el usuario proteja sus datos una vez que quiera cerrar la aplicación, deberá pulsar el botón de salir. El usuario será redirigido a la pantalla principal dejando de estar autenticado.

Mi Sitio

RF3. Editar información personal

Cualquier usuario podrá acceder a esta sección donde tras elegir esta opción, se podrá cambiar cualquiera de sus datos personales.

Agenda

RF5. Añadir evento

Tras acceder a la agenda y seleccionar un día concreto, aparecerá una ventana emergente donde se podrá editar la información del evento. Finalmente se puede guardar esta información o descartarla, en función del botón que se elija.

RF6. Borrar evento

El usuario debe acceder a la agenda y seleccionar un evento existente, tras esto, se le dará la opción de borrar dicho evento, eliminándose de la base de datos.

RF7. Editar evento

Dependiendo de la información de la agenda que se desee cambiar, se debe hacer de una forma u otra. Tras seleccionar la agenda, si el usuario lo que quiere es cambiar la fecha del evento, deberá arrastrar ese evento a la fecha deseada. En cambio, si lo que se desea es cambiar los detalles del evento, al seleccionarlo aparecerá una ventana emergente que permite cambiarlos. Una vez cambiado el usuario deberá pulsar el botón guardar, para guardar los datos en la base de datos.

Buscador de Personal

RF8. Buscar personal

Cualquier usuario puede buscar la información relevante de cualquier miembro del club registrado, mediante su nombre o alguno de sus apellidos. Para ello se ha de acceder a este área mediante el botón de búsqueda de personal.

Mis Equipos

RF9. Registrar entrenamiento

Si el usuario registrado está a cargo de algún equipo, tras acceder a la sección de uno de ellos, tendrá la posibilidad de adjuntar los datos de un entrenamiento realizado.

RF10. Registrar partido

Si el usuario registrado está a cargo de algún equipo, tras acceder a la sección de uno de ellos, tendrá la posibilidad de adjuntar los datos de un partido jugado.

Área Económica

A esta sección tan solo tendrá acceso el Manager, el rol con más permisos posible dentro de la aplicación.

RF12. Registrar ingreso

Dentro del área económica, se podrá registrar un ingreso de dinero en el club, indicando la cuantía y la razón.

RF13. Registrar gasto

Dentro del área económica, se podrá registrar un gasto del club, indicando la cuantía y la razón.

Área Médica

RF14. Añadir jugador lesionado

Cualquier usuario registrado podrá añadir a la lista de jugadores lesionados, el jugador lesionado, la lesión de el mismo y la fecha estimada de recuperación.

RF15. Eliminar jugador lesionado

De la misma manera, cualquier usuario tendrá la capacidad de eliminar un jugador de la lista de jugadores lesionados.

Gestión del Club

A esta sección tan solo tendrá acceso el Manager, el rol con más permisos posible dentro de la aplicación.

RF16. Registrar jugador

El usuario deberá rellenar todos los datos de un nuevo jugador. Este jugador se puede registrar con un equipo asignado o sin el. En caso de no tener inicialmente un equipo asignado, se le podrá asignar en otra funcionalidad.

RF17. Borrar jugador

Se selecciona un jugador tras ser buscado mediante el nombre o mediante el equipo, y se selecciona la opción de borrar jugador. Los datos del jugador se borran de la base de datos.

RF18. Editar jugador

Se selecciona un jugador y el campo de su información que se desee modificar. Tras haber actualizado su información se selecciona la opción de guardar para actualizar los datos en la base de datos.

RF19. Registrar equipo

El usuario rellena la información necesaria de un nuevo equipo y lo guarda, quedando este inicialmente vacío.

RF20. Borrar equipo

Tras buscar el equipo se selecciona y se pulsa la opción de borrar. La información del equipo se borrará de la base de datos, los jugadores de los que estaba compuesto, no, simplemente se borra la asignación que tenían con el equipo.

RF21. Editar equipo

Tras buscar el equipo se selecciona y se actualiza la información que se desee. Después se pulsa guardar para actualizar los datos en la base de datos.

RF22. Registrar personal

Se podrá añadir nuevo personal relacionado con la gestión del club (jugadores no) tras rellenar toda su información y haber indicado su rol. Para que estos nuevos usuario puedan acceder a la aplicación deberán contactar con los responsables de mantenimiento de la aplicación para dar de alta un nuevo usuario.

RF23. Borrar personal

Tras seleccionar una persona dentro del conjunto de personal, se selecciona la opción de borrar. De esta forma la información se borrará de la base de datos y el personal eliminado perderá las credenciales para autenticarse en la aplicación, así como la información que la persona poseía dentro de esta.

RF24. Editar personal

Se selecciona el personal a editar, se actualiza su información y se salva. Este cambio se actualizará automáticamente en base de datos.

RF25. Añadir jugador a equipo

El usuario selecciona un jugador y lo asigna a un equipo. Tanto el jugador como el equipo deben haber sido registrados previamente y el jugador no debe tener un equipo asignado.

RF26. Añadir personal a equipo (entrenado, 2º entrenador, fisioterapeuta..)

El usuario selecciona una persona del personal y lo asigna a un equipo. Tanto la persona como el equipo deben haber sido registrados previamente y en este caso, la persona si puede estar en más de un equipo a la vez.

3.2. Requisitos no funcionales

RNF1. Seguridad y privacidad

Cualquier usuario que quiera acceder a la aplicación deberá autenticarse primero. De esta forma se asegurará que ninguna persona no autorizada, tenga acceso a los datos, ya sean de algún equipo o personales de los usuarios.

RNF2. Mantenibilidad

La aplicación ha sido diseñada para facilitar la introducción de nuevos módulos de forma sencilla y poco costosa. El uso de la arquitectura MVC, que separa la vista de la lógica, facilitará nuevas implementaciones.

RNF3. Portabilidad

Esta diseñada para facilitar la futura portabilidad a dispositivos móviles mediante el uso de invocación de métodos remota, que facilita la tecnología JavaServer Faces.

RNF4. Volumen de datos

Aunque no se prevé un volumen excesivamente grande de datos, no existe un máximo de datos establecido para la aplicación, ni por club, ni por el conjunto de usuarios totales. Es necesario hacer backups con cierta frecuencia.

RNF5. Usabilidad

Se ha procurado que la interfaz de usuario sea lo más sencilla posible, buscando siempre dar facilidades al usuario mediante una navegación intuitiva. Se ha basado la aplicación en la regla de que cualquier punto de la web, debe estar como máximo a 3 clicks de distancia. También se busca minimizar la posibilidad de fallos, acotando el tipo de dato que un usuario puede introducir en la aplicación.

3.3. Ciclo de vida del proyecto

Un punto importante dentro del análisis de un proyecto de software es saber las etapas por las que habrá que pasar para la realización del proyecto. Esto se define mediante la elección de un modelo de vida de software. Existen varias metodologías, pero dadas las características de este proyecto en el que una sola persona se encarga de llevar a cabo todas las fases, evitándose entre otros problemas, los fallos de comunicación, se va a utilizar el modelo de ciclo de vida en cascada.

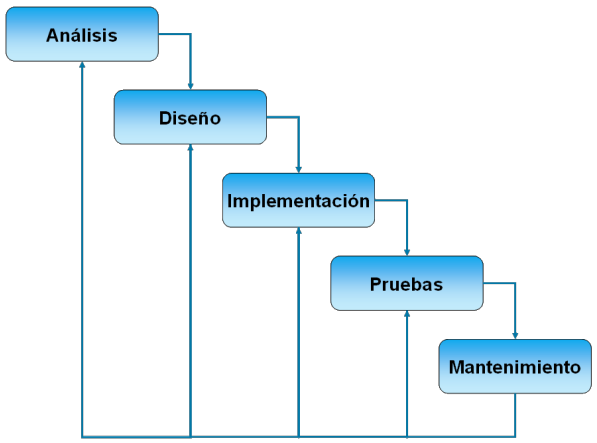

Figura 3: CICLO DE VIDA DEL SOFTWARE

Una de las características mas conocidas de este modelo es que solo cuando acaba una etapa, empieza la siguiente. Debido a que los requisitos se sabían desde el principio, y como se ha comentado antes, solo hay una persona encargada del trabajo, esta característica serviría como guía a la hora de trabajar en el proyecto, de tal forma, que en todo momento se sabría en que paso se está, y que paso es el siguiente. También una vez terminada una fase, se puede volver atrás temporalmente para mejorarse e incluso rediseñarse otra etapa. De esta manera se busca que el funcionamiento de la aplicación sea más eficiente.

3.4. Diagrama de casos de uso

La figura XXX muestra el diagrama de casos de uso de una forma general, teniendo en cuenta los roles de los usuarios. Se intentará representar de una forma clara todas las funcionalidades de la aplicación, así como el tipo de usuario que realiza cada una de ellas.

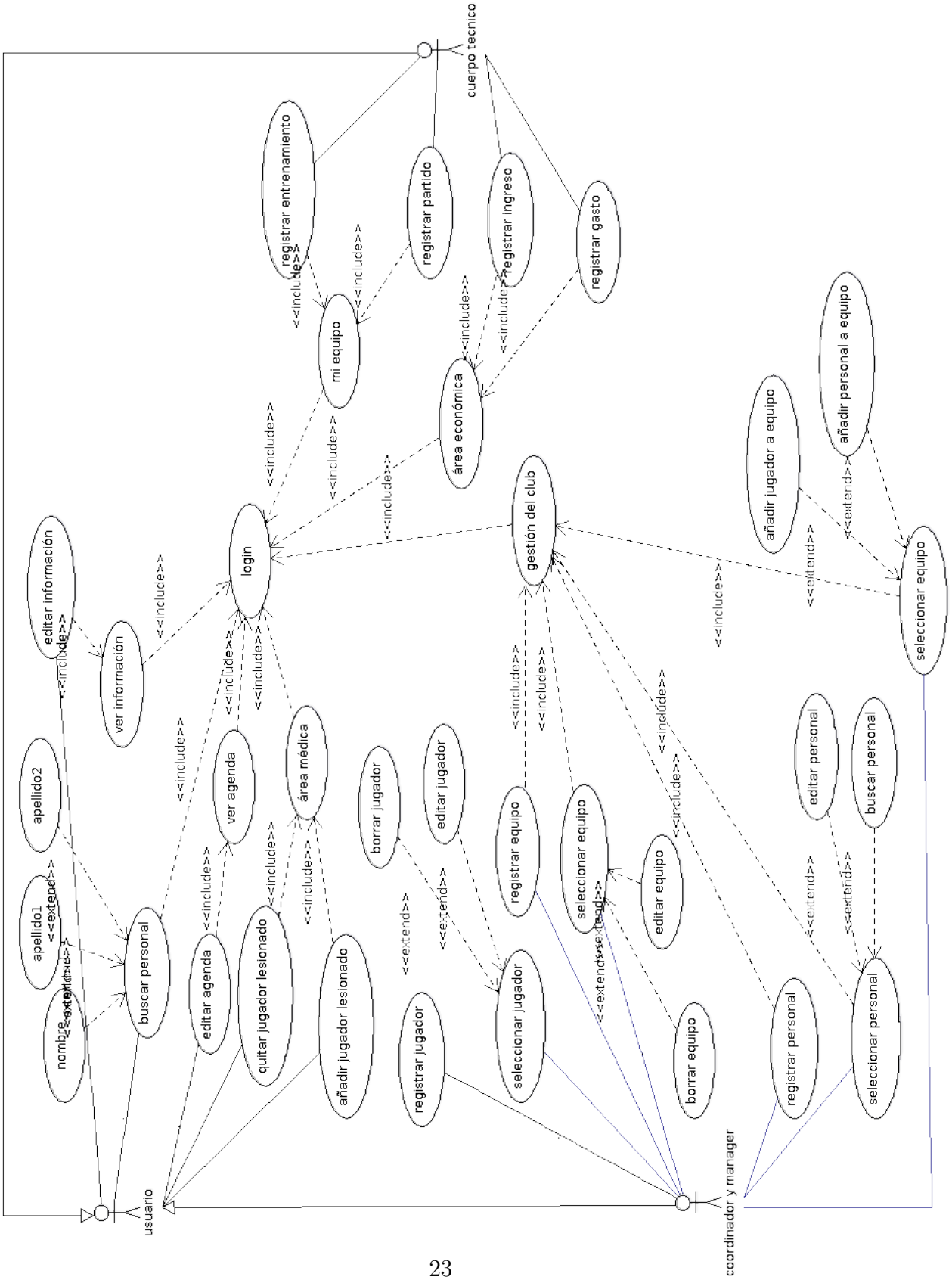

Figura 4: DIAGRAMA DE CASOS DE USO

4. DISEÑO Y DESARROLLO

El diseño es definido en [IEEE610.12 – 90] como "el proceso de definir la arquitectura, componentes, interfaces y las otras características de un sistema o componente". Teniendo esta definición, en este apartado se pretende describir tanto la arquitectura usada, como los componentes y las interfaces y el porque se han tomado estas decisiones.

Dados los requisitos definidos en la etapa anterior, pare la realización de un buen diseño, es conveniente tratar de manera separada cada punto de la definición, y tomar decisiones de forma justificada. Finalmente, esta fase del proyecto nos ayudará a evitar futuros errores causados por un mal diseño de la aplicación, los cuales suelen provocar enormes perdidas de tiempo al tener que volver hacia atrás en el ciclo de vida del software.

El patrón de diseño utilizado en esta aplicación es el Modelo Vista Controlador (MVC). Este patrón se caracteriza por separar conceptos, se debe tener claro en el diseño de la aplicación que la separación de los datos y la interfaz de usuario provocará un código más modularizado, y por tanto, más asequible a la hora de detectar errores. Para ello, divide el software en tres componentes, Modelo, Vista y Controlador.

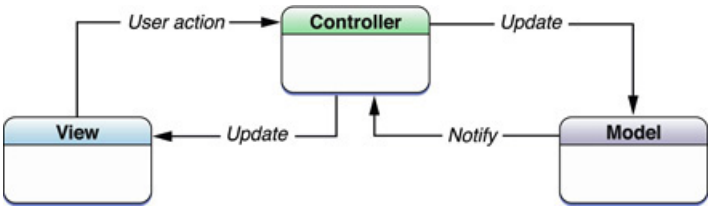

Figura 5: ARQUITECTURA MODELO VISTA CONTROLADOR

El *Modelo* es como se denomina a la lógica de la aplicación, esto va desde el acceso a la información hasta la gestión que la aplicación haga de esta. A este componente, le llega la información del usuario a través del controlador, y una vez procesada, la manda a la Vista. También es la capa donde se lleva a cabo la seguridad de la aplicación y la cual se conecta con la base de datos,

La *Vista* es el componente que interactúa de manera directa con el usuario. Es una interfaz que recoge las acciones de ste y las manda al Modelo, del mismo modo que muestra la información, una vez haya sido tratada.

El *Controlador* este componente recoge las acciones hechas por el usuario en la Vista y las "pasa" al Modelo accionando la funcionalidad correspondiente, del mismo modo,

gestiona los cambios en la Vista producidos por la salida del Modelo. Es el intermediario entre las dos capas anteriores.

4.1. Diseño de la Vista

Como se ha explicado anteriormente, la vista en JavaServer Faces está hecha usando el lenguaje XHTML. Este lenguaje permite incorporar etiquetas propias de JavaServer Faces, produciendo facilidades a la hora de mostrar la información desde la lógica de la aplicación. De esta manera se podrá actualizar un dato automáticamente con el mero hecho de tener una etiqueta que muestre el valor de dicho dato.

En la siguiente imagen, se puede ver como "beanIngreso.usuario.nombre" hace referencia a un dato concreto de la lógica.

```
<h:outputText  value="Nombre: #{beanIngreso.usuario.nombre}" />
<p:commandLink id="editar_nombre" onclick="verForm('nombre_container')"  >Editar</p:commandLink>
<h:form id="nombre_container" style="display: none">
```

Figura 6: ETIQUETA JAVASERVER FACES, MUESTRA LA LÓGICA

Para la realización de la interfaz, se ha hecho uso de la librería Primefaces. Esta librería de componentes JavaServer Faces se usa mediante etiquetas embebidas dentro del código XHTML, de tal forma que el desarrollador se beneficia de componentes y funcionalidades ya hechas. Esta librería tiene como ventajas principales, su simplicidad de uso, y su integración con otros componentes de otras bibliotecas como es el caso de RichFaces.

```
<p:schedule id="schedule" value="#{controladorCalendario.eventosVista}"
            widgetVar="myschedule" timeZone="GMT+2" locale="es">
```

Figura 7: ETIQUETA PRIMEFACES

El calendario de la aplicación, como se puede ver en la siguiente imagen, es un claro ejemplo de la utilidad de Primefaces, estando este ya predefinido.

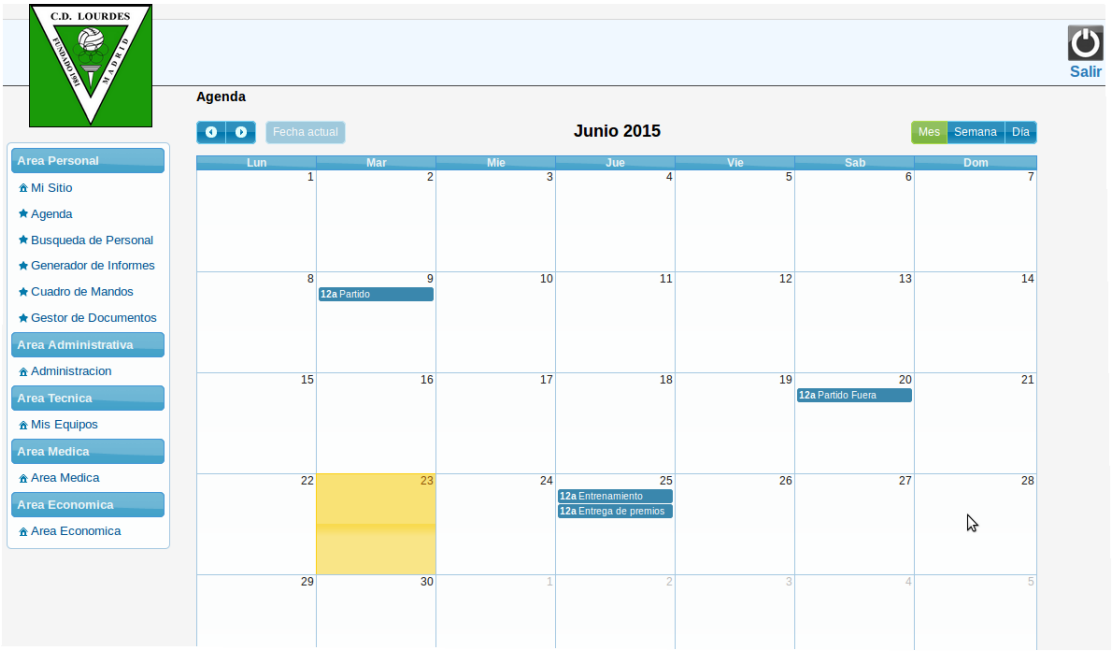

Figura 8: CALENDARIO PRIMEFACES

Como vemos en este ejemplo, las etiquetas que empiezan con 'p' son las referentes a la librería Primefaces.

Finalmente, aunque su uso no es de diseño de interfaces, me gustaría remarcar el uso de la tecnología AJAX en la vista, ya que es aquí donde tiene efecto su funcionamiento. AJAX esta embebido en los ficheros XHTML mediante etiquetas y tiene una utilidad bastante importante. Esta tecnología facilita, más que la creación de aplicaciones web, su uso por parte del usuario, ya que provee de dinamismo a dicha página siendo capaz de cambiar su contenido, sin necesidad de que esta sea recargada por el servidor, Tan solo se solicitan los datos al servidor y se cargan en un segundo plano. Es una forma de meter la lógica en la parte del cliente ayudando a mejorar la funcionalidad de la página, en lugar de que esté toda en la parte del servidor.

En esta aplicación, el uso de AJAX ha tenido lugar en el buscador, y su funcionamiento es el siguiente: Cuando el usuario quiere buscar a una persona del personal, debe escribir en un cuadro de texto su nombre o alguno de sus apellidos, saliendo en una tabla las coincidencias. Para evitar que se tenga que recargar la página por cada búsqueda, y para facilitar la vida al usuario si no sabe los datos completos de la persona que está buscando, la aplicación irá ejecutando querys en la base de datos conforme

el usuario vaya escribiendo, y estos resultados se irán mostrando en una tabla. Para disfrutar de esa agilidad sin necesidad de recargar la página a cada letra que se escriba (algo verdaderamente ineficiente), se hace uso de la tecnología AJAX, permitiendo así la funcinalidad descrita.

A pesar de no poder ver el dinamismo en una foto, en la siguiente imagen se muestra el buscador de la aplicación, el cual esta immplementado usando AJAX. De esta manera, al escribir una sola letra, se mostrarán las coincidencias automáticamente, sin necesidad de recargar la página.

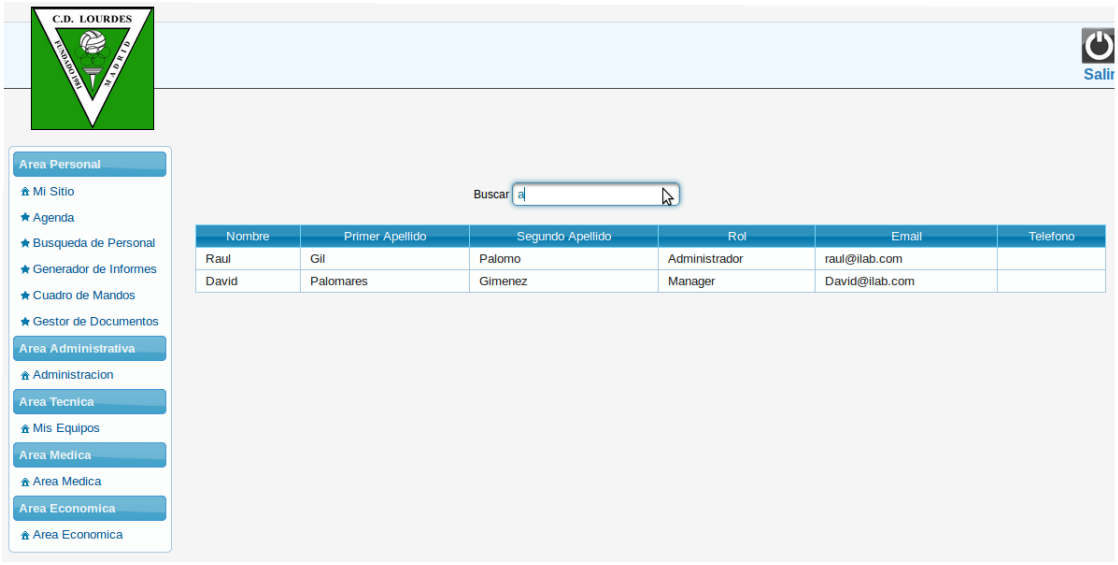

Figura 9: EJEMPLO AJAX

En la siguiente imagen, se puede observar la facilidad de insertar la tecnología AJAX en una etiqueta de la librería Primefaces.

```
<p:ajax event="dateSelect" listener="#{controladorCalendario.onDateSelect}"
        update="eventDetails" oncomplete="PF('eventDialog').show();" />
<p:ajax event="eventSelect" listener="#{controladorCalendario.onEventSelect}"
        update="eventDetails" oncomplete="PF('eventDialog').show();" />
<p:ajax event="eventMove" listener="#{controladorCalendario.onEventMove}"
        update="messages" />
```

Figura 10: EJEMPLO AJAX EN PRIMEFACES

Una parte tan importante como sencilla de la vista, es su estilo. Ha sido quizás una de las partes más amenas de desarrollar en esta aplicación y para llevarla a cabo se ha utilizado CSS. Los ficheros CSS se referencian desde documentos HTML (en este caso

XHTML) de tal modo que en estos CSS se define el estilo deseado de la página, dando valores a los atributos de las etiquetas HTML.

Como punto negativo de este lenguaje hay que destacar que a veces hay problemas para que los documentos HTML referencien a los ficheros CSS, pero hay una gran comunidad, que facilita la resolución de este tipo de problemas.

Para acabar este apartado, solo mencionar el uso de JavaScript para mejorar la interfaz web. Uno de los ejemplos del uso de este lenguaje de programación en esta aplicación, viene de la mano de PrimeFaces. La agenda, como se ha comentado anteriormente, es un elemento de Primefaces. De una forma muy intuitiva se puede crear un *script* en JavaScript con el objetivo de cambiar el idioma de cada etiqueta de la agenda.

4.2. Diseño del Controlador

En cuanto al controlador de la aplicación, la ventaja más destacable que este framework aporta en esta parte de la arquitectura, proviene de la creación de un tipo determinado de objetos, *Managed Beans*, cuyo objetivo es que las páginas JSF (vista) puedan acceder a la lógica de la aplicación. Este tipo de objetos tienen la característica de pertenecer a una clase con un constructor público, sin argumentos, y sus propiedades tienen asociados sus correspondientes métodos get/set, de esta forma se facilita el acceso a sus atributos desde la vista. Las páginas JSF leen los valores de las propiedades del bean que tiene asociada, y cuando se hace un post en un formulario, se guardan sus valores en el bean. No necesitan sen instanciados ya que son inicializados en tiempo de ejecución cuando la aplicación los necesita y como ventaja para los desarrolladores, se facilita su declaración mediante el uso de etiqietas @ManagedBean en vez de tener que declararlos en un fichero de configuración (aunque también se pueden hacer así).

```
@ManagedBean (name="beanIngreso")
@SessionScoped
public class BeanIngreso   implements Serializable{
```

Figura 11: EJEMPLO MANAGED BEAN

En definitiva, los *Managed Beans,* reciben la entrada a través de los formularios, interactúan con el la lógica de la aplicación (modelo en la arquitectura usada) que están implementados por objetos EJB's y genera una nueva vista al usuario, ya sea

cambiando algunos valores de la página o redirigiéndose a otra.

En el siguiente ejemplo vemos como estos *Managed Beans,* muestran la información en la vista. Es absolutamente necesario los métodos *getter* de los atributos a mostrar en la vista para que, automáticamente, se muestren sus valores actualizados.

```
/**
 * Devuelve el nombre actualizado
 * @return nombre_update
 */
public String getNombre_update(){

    return nombre_update;
}


/**
 * Devuelve la contrasenna actualizada1
 * @return pwd_update1
 */
public String getPwd_update1(){

    return pwd_update1;
}
```

Figura 12: EJEMPLO 2 MANAGED BEAN

Finalmente, en esta última imagen podemos ver claramente como este método llama a la lógica de la aplicación, y en función del valor que este le devuelva, el controlador mostrará en la aplicación un mensaje u otro.

```
public void cambiaMail(){

    FacesMessage msg = new FacesMessage();
    msg.setSeverity(FacesMessage.SEVERITY_ERROR);
    if(mail_update.length()==0){
        msg.setSummary("¡Error!, campos sin completar");
        FacesContext.getCurrentInstance().addMessage(null, msg);
        mail_update="";
        return;
    }
    dao.actualizaMail(mail_update,usuario.getId());
    usuario.setMail(mail_update);
    mail_update="";
    msg.setSeverity(FacesMessage.SEVERITY_INFO);
    msg.setSummary("E-Mail actualizado");
    FacesContext.getCurrentInstance().addMessage(null, msg);
    return;

}
```

Figura 13: EJEMPLO 3 MANAGED BEAN

4.3. Diseño del Modelo

Por último, la mayor ventaja de esta parte de la arquitectura no viene dada por una característica propia del framework, sino por una ventaja que aporta una API del estándar Java EE, los Enterprise Java Beans. Estos objetos son provistos desde el lado del servidor en la aplicación, y tienen bastantes ventajas para el desarrollador, ya que les permite centrarse únicamente en la creación de la lógica sin necesidad de preocuparse por temas como la concurrencia o la seguridad que vienen resueltos de serie y pueden provocar errores en el resultado final, ante un mal diseño de la aplicación. Pero de todas estas características, su mayor virtud es que soportan invocación remota (RMI). Esto hace a los Enterprise Java Beans una opción muy atractiva a la hora de diseñar una aplicación, con vistas a una futura ampliación a otros dispositivos, aunque en esta implementación del proyecto solo se ha llevado a cabo la implementación de la interfaz local. Existen tres tipos de Enterprise Java Beans:

- ENTITY EJBS: Encapsulan objetos que almacenan datos del lado del servidor. A partir de Java EE5.0, estos beans desaparecen.

- SESSION EJBS: Representa un proceso o una acción de la lógica, gestionan el flujo de información en el servidor, pero esta información se mantendrá solo durante el tiempo que el cliente interactua con el bean. Normalmente, cualquier llamada a

un servicio del servidor debería comenzar con una llamada bean de sesión. No se comparten entre más de un cliente. Hay dos tipos:

Stateful: Las variables del bean almacenan datos obtenidos durante la conexión del cliente. Cada bean de sesión Stateful, por tanto, almacena el estado de un cliente que interactúa con el bean. Este estado se modifica conforme el cliente va realizando llamadas a los métodos de negocio del bean. El estado no se guarda cuando el cliente termina la sesión. Un ejemplo típico es un carrito de la compra, donde el cliente va guardando los ítem que va comprando.

Stateless: Estos beans no se modifican con las llamadas de los clientes. Los métodos con los que interactúa el usuario en este tipo de beans reciben datos y devuelven resultados, pero no modifican internamente el estado del bean. Esto permite que se cree un único bean, que pueda estar asignado a múltiples clientes, ya que la asignación solo dura el tiempo de invocación del método solicitado. Este tipo de beans se usan para encapsular procesos de negocio más que datos de negocio, se usan para tareas que no están ligadas a un cliente específico, o como puente de acceso a una base de datos. Para las arquitecturas MVC, el bean de sesión podrá proporcionar a la interfaz de usuario del cliente los datos necesarios, así como modificar objetos de negocio (base de datos). Un ejemplo típico es una aplicación que calcule el valor que debe pagar un cliente en función de los datos de dicho cliente pasando como argumento del método correspondiente el conjunto de items que el cliente ha comprado.

- Menssage-driven EJBs: Permiten que se reciban mensajes JMS(Java Messaging system) de forma asíncrona, de esta manera, el hilo de ejecución de un cliente no se bloqueará mientras espera que se complete algún método.

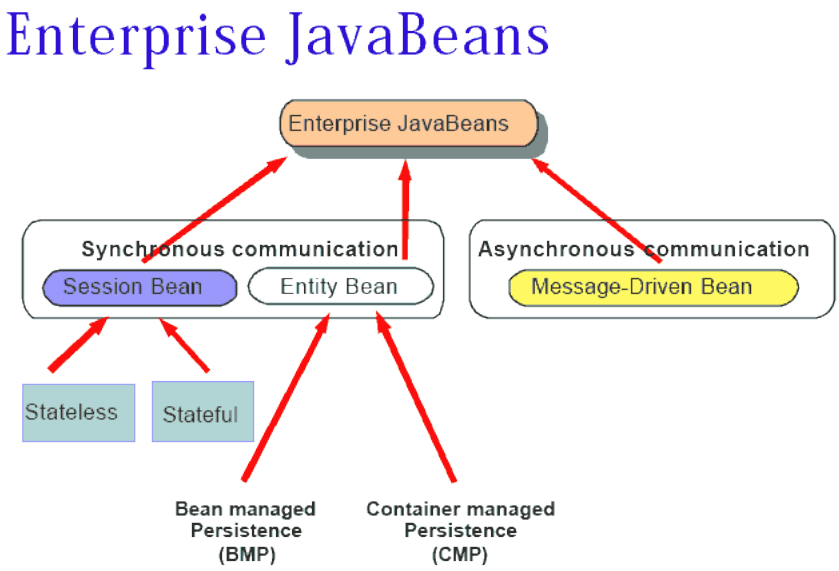

Figura 14: TIPOS DE EJB

Para el desarrollo de la lógica de este proyecto, se han usado SESSION EJBs, concretamente STATELESS beans. Para ello, simplemente se ha de indicar el tipo de bean mediante una etiqueta.

```
@Stateless
public class EquipoDaoBean extends DBTester implements EquipoDaoBeanLocal {
```

Figura 15: SESSIONEJBs

Me he decantado por SESSION EJBs en lugar de MENSSAGE-DRIVEN EJBs, ya que el modelo de la aplicación debe soportar funcionamiento síncrono. Dentro de los SESSION EJBs se eligen para los objetos de la lógica los STATELESS beans en lugar de STATEFUL beans (se usan para encapsular procesos de negocio) ya que las operaciones a realizar son operaciones con la base de datos, en cambio, como he comentado anteriormente, en el caso de los STATEFUL beans, el estado no se guarda al terminar la sesión.

En la siguiente figura se puede ver como se declara un objeto SESSION EJBs, y como posteriormente es usado en una llamada a base de datos.

```
@EJB UsuarioDaoBeanLocal dao;
  •
  •
  •
dao.actualizaPwd(pwd_update1,usuario.getId());
```

Figura 16: EJEMPLO SESSION EJBs

4.4. Diseño de la base de datos

En este apartado se mostrará la estructura de la base de datos usada en la aplicación. La mayoría de las tablas están relacionadas con alguna otra tabla, por lo que, para modelar este problema, se usa un modelo relacional de datos.

La base de datos consta de 17 tablas donde queda almacenada toda la información relativa a la aplicación, a continuación se muestra como se ha diseñado la estructura de esta base de datos.

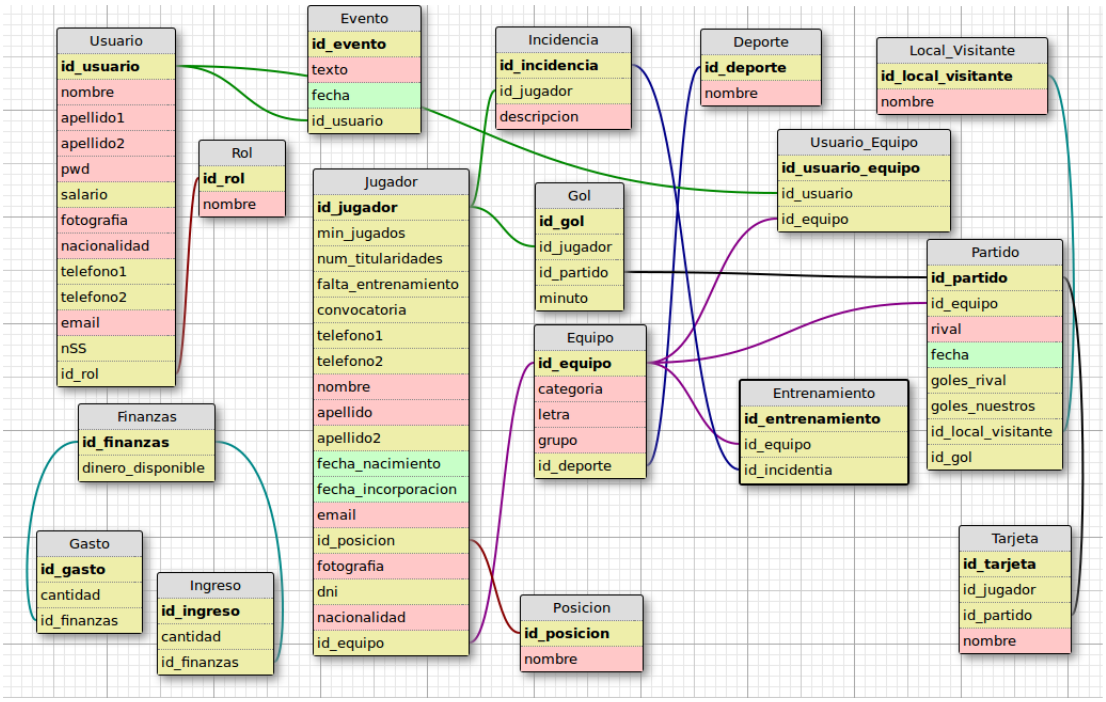

Figura 17: ESQUEMA RELACIONAL

Como se ha explicado anteriormente, la conexión de la base de datos con la lógica de la aplicación se lleva a cabo mediante los Enterprise Java Beans, concretamente

mediante Stateless. Estos Beans tienen un conjunto de métodos con los cuales acceden a esta base de datos, ya sea para obtener información, para añadirla o para actualizarla. Estos métodos se acceden a la base de datos mediante el uso de la API JDBC (Java Data Base Connectivity), más específicamente, usando un "Connection Pool", el cual permite tener varias conexiones que pueden ser reutilizadas por los diferentes usuarios, evitando así el coste de tener que abrir y cerrar cada conexión que use el usuario. Este pool está administrado por un servidor de aplicaciones (GlassFish) que va asignando las conexiones a medida que los clientes van solicitando el acceso a la base de datos.

Se ha elegido esta API ya que las operaciones sobre la base de datos se llevan a cabo en Java.

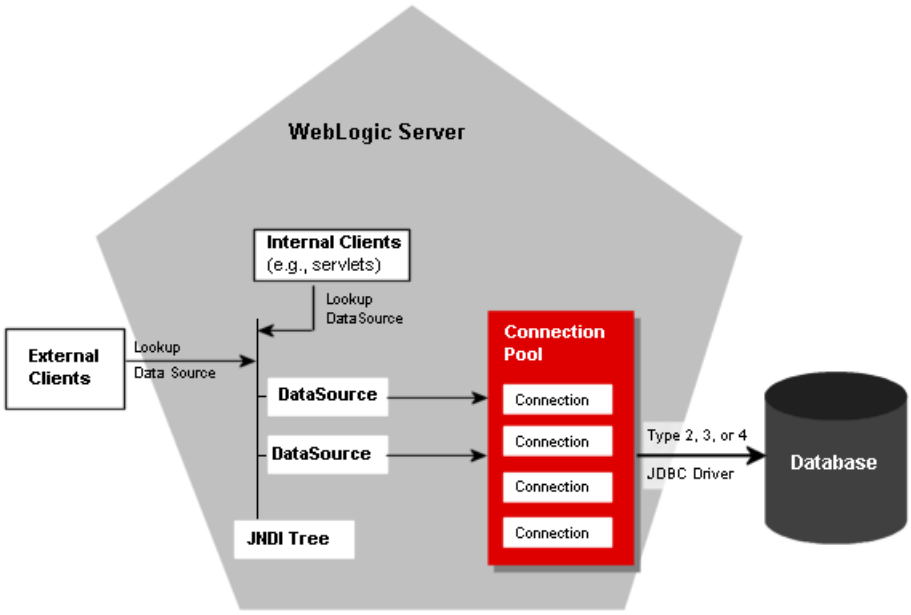

Figura 18: ARQUITECTURA CONNECTION POOL

Para gestionar la base de datos se usará el gestor PostgreSQL, entre otras cosas porque es un gestor libre, de los más conocidos, y por consiguiente con una gran comunidad en internet, que resulta de gran ayuda en temas como la configuración del gestor.

Opción Java EE (JPA)

A pesar de haber implementado la base de datos de la manera descrita, la plataforma Java EE cuenta con una API que facilita el manejo de bases de datos. Se trata de Java

Persistence API (JPA), consiste en un framework que maneja datos relacionales en aplicaciones, de tal forma que no pierde las ventajas de la orientación a objetos al interactuar con una base de datos.

La característica más representativa de este framework son las Entidades (Entity), son los objetos que instancian las bases de datos, siendo una Entidad una tabla en la base de datos relacional, y cada instancia de la entidad, una fila de la tabla. Una gran ventaja de JPA, es la facilidad para que una clase Java instancie una tabla de una base de datos, simplemente se han de seguir algunas reglas como la declaración específica del constructor o el etiquetado de métodos de una manera determinada, entre otras. No se pretende definir aquí el funcionamiento exacto de la API, por eso no se entrará más en detalle acerca de los requisitos de esta.

Las ventajas más importantes que presenta la JPA respecto a la orientación a objetos son la herencia y el polimorfismo. Gracias a la herencia, clases Entidad, pueden heredar de clases normales y viceversa.

Respecto al lenguaje de querys, Java proporciona un lenguaje llamado Java Persistence Query Languaje (JPQL). Se trata de un lenguaje similar a SQL que funciona independientemente del sistema de gestión de base de datos. También existe otra forma de llevar a cabo las querys, y es mediante la API, "Criteria". Usando Criteria, las querys son escritas en Java, es similar a JPQL y también funciona independientemente del sistema de gestión de base de datos. Aunque las querys hechas en JPQL son quizás más intuitivas, debido a su semejanza con SQL, Criteria puede llevar a menos errores, ya que al estar escrito en Java y no requiere el uso de casting (al contrario que JPQL).

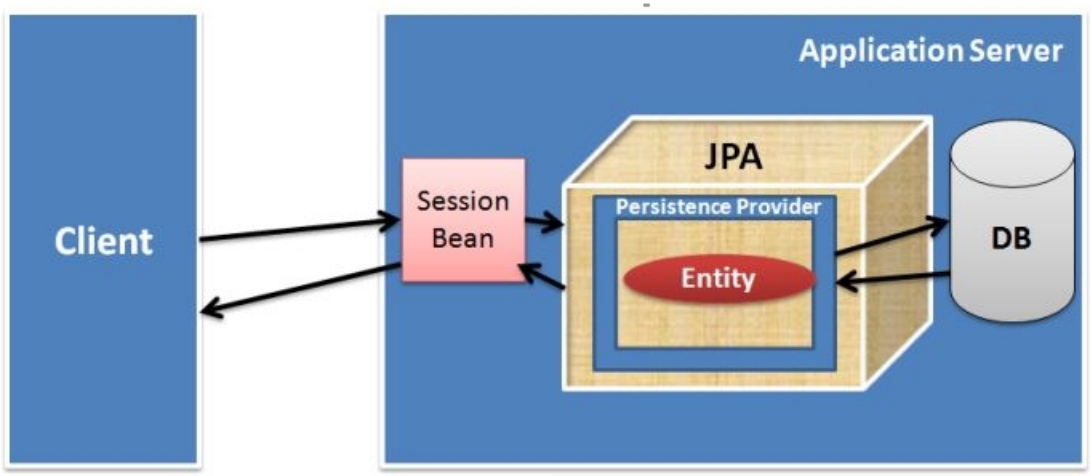

Figura 19: ARQUITECTURA JPA

5. PRUEBAS Y RESULTADOS

Es la cuarta etapa del modelo de ciclo de vida en cascada, durante la cual se verifica el correcto funcionamiento de software desarrollado o se buscan posibles errores para solventarlos. Se buscarán condiciones o partes de la aplicación donde se crea que esta puede tener un comportamiento diferente al esperado, algunos casos factibles de error podrían ser:

- En las entradas a la aplicación.
- Tiempo de respuesta.
- Funciona en diferentes dispositivos.
- El resultado encaja con lo esperado.

Como podemos observar, esta etapa engloba a los requisitos funcionales y no funcionales descritos en el apartado *3 Análisis*, y será en este apartado donde se procederá a documentar los casos probados.

Las pruebas realizadas se han dividido según sus tipos y se han llevado a cabo siguiendo un orden lógico, primero las pruebas más a bajo nivel, para ir pasando cada vez a un nivel más alto de abstracción. Estos son los tipos de pruebas realizados:

1. Pruebas unitarias: Este tipo de pruebas se llevan a cabo en los diferentes módulos que componen el proyecto, comprobando su correcto funcionamiento por separado. Se diseñará cada caso de prueba sin tener en cuenta el funcionamiento de otras funciones. El hecho de llevar a cabo estas pruebas en primer lugar, facilitan que los posibles fallos que tengan lugar en la integración, no resulten del mal funcionamiento de un módulo.

2. Pruebas de integración: Estas pruebas son realizadas después de cerciorarse que los módulos funcionan correctamente por separado. Se comprueba que estos módulos, una vez se hayan juntado para llevar a cabo una funcionalidad, funcionen juntos correctamente.

3. Pruebas de validación: Este conjunto de pruebas se lleva a cabo en último lugar. Simplemente comprueba que la funcionalidad de la aplicación, es decir, de todos los módulos funcionando de manera integrada, sea la requerida, especificada en este caso en los requisitos .

En las siguientes páginas, se mostrarán un conjunto de pruebas realizadas. Debido a que tanto las pruebas unitarias como las de integración muestran el correcto funcionamiento de métodos propios del código, y en esta memoria no se ha hablado del código implementado, sino que se muestra la aplicación desde un nivel más alto, se ha optado por hacer una muestra de algunas de las pruebas de validación realizadas. Estos son los algunos ejemplos del plan de pruebas realizado:

Nombre	Descripción	Salida Esperada
Inicio de sesión	El usuario se identifica con su nombre y su contraseña	El usuario accede a la aplicación
Inicio de sesión erroneo	El usuario se identifica con un nombre y/o contraseña erróneo	El usuario permanece en la ventana de log-in mostrándose un mensaje de error
Actualización del nombre	El usuario edita su nombre tras acceder a "Mis Datos", dentro de "Mi Sitio"	Se muestra un mensaje donde se indica que se ha actualizado el nombre correctamente
Actualización fallida de la contraseña	El usuario edita su contraseña tras acceder a "Mis Datos" dentro de "Mi Sitio", escribiendo la nueva contraseña dos veces de forma diferente	Se muestra un mensaje donde se indica que se ha habido un error al escribir la nueva contraseña
Añadir un evento	El usuario selecciona un día de su agenda y crea un evento	Se muestra la agenda con el evento añadido en el día seleccionado
Borrar un evento	El usuario selecciona un evento de su agenda y lo borra	Se muestra la agenda sin el evento borrado
Editar fecha evento	El usuario selecciona un evento de su agenda y lo arrastra hacia otro día	Se muestra la agenda con el evento en su nuevo día
Buscar personal	El usuario accede a "Búsqueda de Personal" y escribe el nombre y/o apellidos de la persona a buscar, existrendo coincidencias	Se muestra en una tabla el o los usuarios buscados con sus datos más relevantes
Búsqueda de personal errónea	El usuario accede a "Búsqueda de Personal" y escribe el nombre y/o apellidos de la persona a buscar, sin que exista ninguna coincidencia	Se muestra en una tabla un mensaje indicando que no se encontraron usuarios

Todas la pruebas que aquí se muestran han sido superadas satisfactoriamente y la salida esperada encaja con los requisitos descritos anteriormente.

En la siguiente imagen se muestra el resultado de la segunda prueba, el usuario permanece en la ventana de log-in mostrándose un mensaje de error.

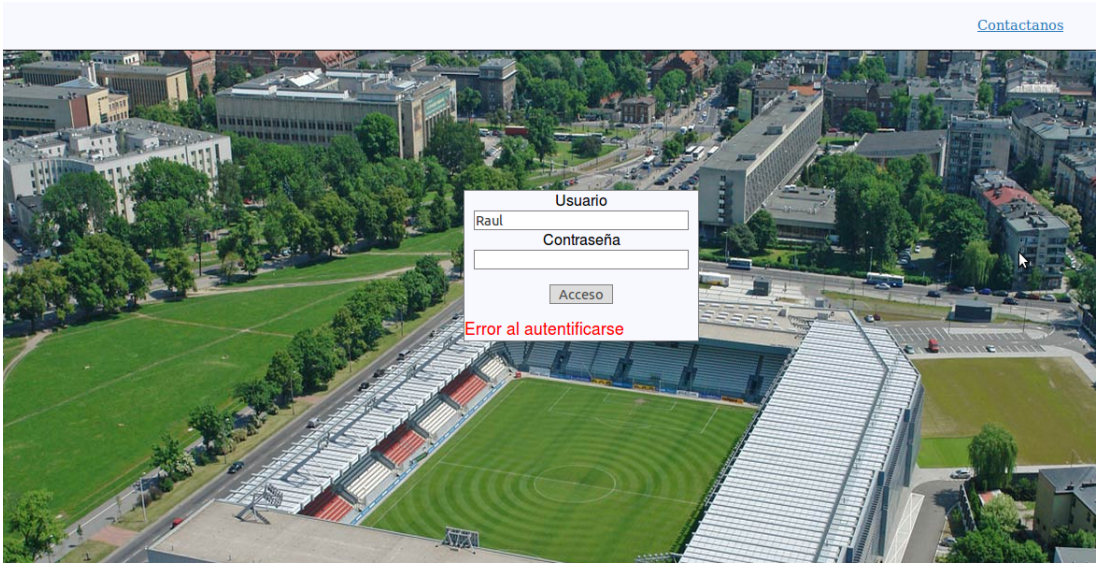

Figura 20: INICIO SESIÓN ERRONEO

Otro ejemplo de una prueba realizada, en este caso la descrita en la tabla en la fila número 4, se puede ver en la siguiente imagen. Se muestra un mensaje donde se indica que se ha habido un error al escribir la nueva contraseña.

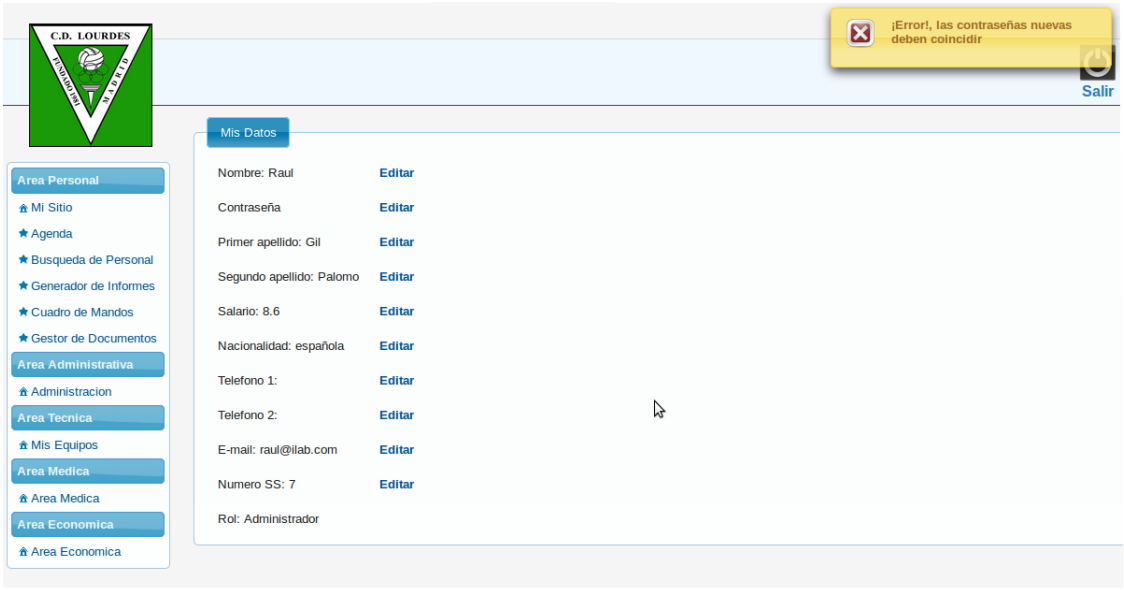

Figura 21: ACTUALIZACIÓN FALLIDA DE LA CONTRASEÑA

Me gustaría destacar la gran utilidad de JUnit para realizar pruebas unitarias. Este framework es estudiado en asignaturas como Proyecto de Análisis y Diseño de Software y se ha aprovechado el uso de Java para beneficiarse de sus ventajas, sobretodo su facilidad de uso y su adaptabilidad al entorno de desarrollo.

6. INCREMENTOS FUTUROS Y CONCLUSIONES

En este apartado primero se hablará sobre los incrementos que se plantean hacer en el futuro para esta aplicación, de tal forma que se mejore el producto con respecto a lo que hay actualmente. También se busca exponer las conclusiones sacadas del mismo trabajo ya sean conceptos aprendidos o reflexiones.

6.1. Incrementos futuros

Tras el análisis y diseño realizado hasta ahora, existen unos cuantos temas abiertos que deberían ser mejorados en el futuro con el objetivo de hacer una aplicación más atractiva, así como extenderla a otros dispositivos.

El primer tema que debería ser tratado es el gestor de informes. Sería de gran utilidad un gestor de documentos, el cual pudiera recibir documentos en formato Excel y cuya información fuese directamente guardada a la base de datos de la aplicación. De esta manera, los clubes podrían sincronizar toda su información (en caso de que esta existiese) con la página, sin necesidad de ir rellenando dato a dato. Se ha pensado en este formato Excel ya que es el editor que más se usa en este tipo de entornos para guardar la información.

Otro futuro incremento que debe ser realizado es la mejora de la parte visual de la aplicación. Aunque su aspecto actual es bueno, quizás el uso de tecnologías como HTML 5 y CSS 3, podrían darle a la aplicación un aspecto más modernista, mejorando así la motivación por parte de los usuarios a usar la aplicación. En definitiva, hacer la web más atractiva.

El incremento más interesante que se tiene en mente es sin ninguna duda la realización de una aplicación móvil. Debido a la forma en que se ha desarrollado su aplicación, a su diseño y a la elección de las tecnologías empleadas, gran parte del trabajo de este incremento ya estaría hecho.

Como se ha explicado a lo largo de esta práctica, gracias a los EJB, algunos métodos de la parte de la lógica podrían ser reutilizados. De esta manera, habría que centrarse tan solo en desarrollar la interfaz y el controlador.

Para llevar a cabo la reutilización del modelo tan solo haría falta implementar la interfaz remota de los EJBs usados. En cuanto al acceso de estas interfaces remotas, habría varias posibilidades:

- RMI-IIOP: Es un estándar que permite que diversos componentes software desa-

rrollados en diferentes lenguajes de programación y ejecutándose en diferentes dispositivos, puedan trabajar juntos. Usa la interfaz RMI de Java en lugar del sistema CORBA aunque está basado en la misma especificación que esta.

- SOAP: Es un protocolo que define cómo dos objetos en diferentes procesos pueden comunicarse por medio de intercambio de datos XML. En el caso de la aplicación, los métodos de los EJBs podrían ser exportados como servicios web basado en este protocolo. La plataforma Java EE facilita su uso, ya que con la simple etiqueta @Webservice en un EJBs, sus métodos pueden ser invocados por el cliente.

Finalmente, otro punto que debe ser mejorado es el servidor. Actualmente el servidor solo funciona en local, donde se han llevado a cabo las pruebas. Como es lógico, para que la aplicación pueda ser comercializada, habría que migrarla a un servidor con servicio de hosting.

6.2. Conclusión

En este apartado se recogerán tanto las conclusiones técnicas obtenidas a partir de la realización de este proyecto, como las experiencias personales extraídas a lo largo del desarrollo de este.

A pesar de faltar algunos requisitos por terminar de desarrollar, la base de este proyecto, se ha realizado satisfactoriamente. Antes de empezarlo se buscaba desarrollar una aplicación web que pudiera ayudar en el ámbito del fútbol base, y sin lugar a dudas, se puede decir que a grosso modo se han cumplido los objetivos.

También, ha sido satisfactorio haber llevado a cabo lo aprendido en asignaturas como Proyecto de Análisis y Diseño de Software, Sistemas Informáticos, Estructuras de Datos o Ingeniería del Software de manera individual. Es gratificante ver como lo estudiado durante la carrera lo he sabido plasmar en un proyecto tal y como se hace en el mundo real, sin las facilidades que se nos dan en las prácticas de las asignaturas.

También, este trabajo me ha enseñado bastantes conceptos nuevos que desconocía. Para empezar, estudié a fondo la documentación que proporciona Oracle en su web acerca de la plataforma de programación Java EE. A pesar de que Java es un lenguaje estudiado en detalle a lo largo de diferentes asignaturas de la carrera, existen una infinidad de conceptos de los que no era conocedor, y este tutorial los explica de manera precisa. Del mismo modo, para hacer la comparativa de frameworks, tuve que leer minuciosamente diferentes títulos sobre cada uno de ellos, aprendiendo sus características,

y viendo en que circunstancias, el uso de unos era mejor que el uso de otros.

No hay que dejar de lado la base de datos. A pesar de haber usado el Pool de conexiones, algo que es relativamente eficiente ya que reutiliza conexiones con la base de datos, pienso que sería más eficiente y fácil de desarrollar para futuras ampliaciones en la base de datos, el uso de Java Persistence API.

Finalmente, en el ámbito personal, el haber desarrollado esta aplicación creo que me va a aportar mucho. He visto que soy capaz de llevar a cabo de principio a fin, una idea gracias a los conocimientos adquiridos en estos años, y en un mundo donde las aplicaciones web están a la orden del día, saber plasmar desde cero, una imagen mental en una aplicación es algo realmente gratificante.

Referencias

[1] Ruby on rails, Bruce Tate, Anaya Multimedia, 2007,

[2] La guía definitiva de Django, Adrian Holovaty, Anaya Multimedia, 2009

[3] Cocoon 2 programming web publishing with XML and Java, Bill Brogden, Sybex, 2003

[4] JavaServer Faces 2.0 the complete reference, Ed Burns, McGraw-Hill, 2009

[5] Tecnologías asp.net 4.0 saltando desde la versión 2.0, José Manuel Alarcón Aguín, Krasis, 2009

[6] Professional ASP.NET 2.0, Bill Evjen, Anaya Multimedia, 2007

[7] The Definitive guide to symfony, Zaninotto, François, Apress, 2007

[8] Arquitectura Cliente-Srvidor, http://eltamiz.com/elcedazo/2010/06/24/sistemas-cliente-servidor-vs-sistemas-multi-capa/

[9] Arquitectura dirigida por eventos, http://c2.com/cgi/wiki?EventDrivenProgramming

[10] Modelo Vista Controlador, http://martinfowler.com/eaaDev/uiArchs.html

[11] Definición de diseño de software, http://ieee-elearning.org/course/search.php?search=computer&perpage=10&page=4

[12] Comparativa JSF y Spring MVC, http://www.arquitecturajava.com/categoria/spring/spring-mvc/

[13] Ejb, http://www.jtech.ua.es/j2ee/2003-2004/abierto-j2ee-2003-2004/ejb/sesion01-apuntes.htm

[14] Ejb, http://docs.oracle.com/javaee/7/tutorial

[15] Conexion pool, http://www.juntadeandalucia.es/servicios/madeja/contenido/recurso/218

[16] Java Persistence API, http://docs.oracle.com/javaee/6/tutorial/doc/bnbpz.html